DATA TRANSFORMATION FOR THE REDUCTION OF POWER AND NOISE IN CMOS STRUCTURES

Claim for Priority:

The subject patent application hereby claims priority from U.S. Provisional Patent Application Serial No. 60/277,587 (Hagen), filed 21 March 2001.

Background of the Invention

In large CMOS logic structures, the primary source of power dissipation and noise results from the transition of logic levels throughout the structure. This power dissipation and noise can be a significant limiting factor in the implementation of large CMOS logic structures. It is noted that in CMOS circuitry, power dissipation is directly proportional to the number of bit transitions propagating through the structure.

What is needed is a way to reduce the number of logic transitions that occur in a pipelined structure while preserving the information in the data.

Summary of the Invention

A data transformation algorithm is selectively applied to each data vector as it enters the pipelined structure. In a selection step, the algorithm compares the bit value of the new data vector with the corresponding bit values of the preceding data vector, and sums the number of logic transitions. The transformation algorithm is applied to the new data vector only if it would reduce the resulting number of transitions, otherwise the data vector is propagated unmodified.

Bit inversion is a data transformation algorithm according to the present invention that provides a reduction in the number of logic transitions of up to 50%.

Brief Description of the Drawing

FIGURE 1 shows a table useful for understanding the subject invention.
FIGURE 2 shows a pipelined logic transformation structure, in block diagram form, according to the subject invention.
FIGURE 3 shows apparatus, in block diagram form, useful for implementing the subject invention.
FIGURE 4 shows a table useful for understanding the subject invention.

Detailed Description of the Embodiment

CMOS (Complementary Metal-Oxide Semiconductor) technologies provide the capability to design very large digital structures. The main contributor to power dissipation in CMOS circuits is the transition of logic states within the logic gates. The power dissipated in this way is proportional to the number of gates that produce a logic transition per unit time. At high clock rates, this dynamic power dissipation can be the limiting factor for large CMOS designs.

One method for reducing the dynamic power dissipation in CMOS components is to employ architectures that reduce the number of logic transitions in the implementation of a desired function. An example of this approach is the use of gray code instead of binary code for sequential address generators. Gray code produces only a single logic transition for each sequential change of state, whereas, on average, binary code will produce one-half as many logic transitions as there are bits in the code.

Statistically, for an N-bit code, one can express the total number of states wherein at least one bit changes from a previous state as $2({}_2C_{2^N})$ possible outcomes. That is, the parenthetical expression $({}_2C_{2^N})$ defines the total number of unique combinations of two sequential data words from an entire set of data words. The multiplier of 2 accounts for a positive going changes and negative going changes (i.e., a value of 7 falling to a value of 4, and a value of 4 rising to a value of 7 should both be counted). For a 3 bit code, N=3 and

$2({}_2C_{2^3}) = 2\frac{8!}{2\times 6!} = \frac{8\times 7\times 6!}{6!} = 56$ possible outcomes having a bit transition.

All possible transitions from one 3-bit value to another 3-bit value are shown in FIGURE 1. There are fifty-six possible transitions in which at least one bit will change from the previous state to the new state, and eight possible "transitions" in which no bit changes at all, for a total of sixty-four possible outcomes. If one were to add all of the logic transitions (total number of bits that change) in these sixty-four possible outcomes, the result would be ninety-six bit-changes. The average of ninety-six bit-changes over 64 possible outcomes is 1.5. In accordance with the relationship given above, in a binary code having only 3 bits the average number of transitions equals 1.5, or one-half the number of bits in the code.

It is herein recognized that use of an architecture that reduces the number of logic transitions for a given function can dramatically reduce the power consumed in CMOS circuits.

The goal of the approach of the subject invention is to reduce the number of logic transitions produced in a pipelined logic structure. As a sequence of new data vectors is presented to such a structure, each new vector will produce a number of logic transitions. The number of logic transitions depends upon the prior state of the inputs. Once the data enters the pipeline, these transitions repeat on each clock edge until the data exits the structure. For many pipeline structures, the data is directly operated on in relatively few pipeline locations when one considers the depth of the pipeline.

In these cases, a power-saving advantage can be gained by transforming the data vector into a new value which will generate fewer logic transitions as it propagates through the pipeline. At locations where the data must be operated on, the inverse transformation is performed, thus regenerating the original data vector. By this method, the total number of logic transitions is reduced in the pipeline structure.

FIGURE 2 shows an example of a pipelined logic transformation structure in accordance with the subject invention. Data vectors enter the structure at point **201**, and a data vector that is one sample older in time exists at point **207**. These two vector are presented to controller **210** which determines the transformation algorithm to be employed, and produces a resulting transformation vector at point **212**. This transformation vector is coupled to a sampler unit **205** and applied to the incoming data vector at point **201** when it is sampled by sampler unit **205**. The newly transformed data vector at point 207 and the transformation vector at point 212 are propagated in parallel through the pipeline structure **220**, **230**, **240** and **260**, **270**, **280**. For simplicity and ease of explanation, pipeline structure **220**, **230**, **240** and **260**, **270**, **280** represented as a series of latches.

One skilled in the art will realize that each stage of pipeline structure **220**, **230**, **240** and **260**, **270**, **280** may be a stage in which some data processing may occur. In this regard, assume that the original data vector is to be operated on by a stage (not shown) following point **290**. In such a case the data vector may need to be restored to its original form before such processing occurs. Restoring the data vector to its original form is accomplished in a data transformation unit **250** by performing a reverse transformation.

The following facts should be considered when evaluating whether to implement the logic transformation arrangement of the subject invention. To implement such a logic transformation arrangement, additional logic circuitry must be provided. The transformation vector must be propagated in parallel with the vector data, thus increasing the pipeline width. Propagating the transformation vector through the pipeline increases the number of logic transitions that occur. Reverse transformation circuitry must be provided to recover the data in its original form. The approach works best when the transformation and reverse transformation algorithms are easy to implement, the transformation vector is simple (i.e., has few bits) and the actual data (i.e., the data in its original form) is needed at only a few points in the pipeline.

A more detailed view of an embodiment of the invention is shown in FIGURE 3. In this example the selected data stream width is 11 bits. This width was selected based on tradeoffs between three parameters of the circuit. A wider data vector produces logic transitions closer to the theoretical limit of 50%, wider data vectors increase the complexity and delay of the summation circuitry, and an odd number of bits in the data vector produces a symmetry which improves the efficiency of the transformation algorithm.

In this embodiment, the transformation algorithm employed is vector inversion. As 11-bit data enters the circuitry 300 at an input point 301, a determination is made of how many bits will change (i.e., number of logic transitions) from the previous sample. This is accomplished by performing an exclusive-OR (XOR) function on each bit position of the new data vector and its counterpart bit position in the preceding data vector. Data from the Q output of latch 305 (the new data vector) is applied to one input of an XOR-gate 320. Data from the Q output of latch 315 is applied to the other input of XOR-gate 315. In accordance with the normal rules for the operation of an XOR function, a "logical true" output signal will be produced only when the two inputs are in opposite states. Thus, each "logical true" signal represents one bit transition. The output of XOR gate 320 is applied to a summer unit 325. Summer unit 325 adds all of the "logical true" signals that it receives and applies the sum to a controller 330.

Controller **330** determines if the sum of the transitions exceeds one-half of the number of bits in the code. In this case, 11 divided by 2 yields 5.5 transitions. Therefore, 6 transitions, or more, will cause controller **330** to initiate a data transformation to reduce the number of transitions in the data vector. In operation controller **330**, produces a logic level signal of the proper polarity to cause a 2-to-1 multiplexor (MUX) **310** to select the data at input A if it is determined that no data inversion is required, or to select the data at input B if it is determined that a data inversion is required. Note that the data at the Q output (non-inverting output) of latch **305** is applied to input A of MUX **310**, and the data at the Q-BAR output (inverting output) of latch **305** is applied to input B of MUX **310**. Thus, the selected data output signal of MUX **310** will be latched into latch **315** and propagated through pipeline elements **340**, **370**. The transformation vector from controller **330** will be propagated through pipeline elements **335**, **375** in parallel with its corresponding data vector. A reverse transformation is preformed in a 2-to-1 MUX **380** under control of a selection signal generated by latch **375**. As can be readily seen, this selection signal is the transformation vector corresponding to that particular data vector.

To evaluate the benefits of the transformation algorithm, we first calculate the performance of the circuit without transformation. When random data is applied to the circuit, it can be shown that the average number of logic transformations produced by each vector is 5.5 (one-half of 11 as noted above). The worst case state changes produce 11 bit transitions (00000000000 → 11111111111 and 11111111111 → 00000000000). When inversion is applied, the worst case patterns are transformed respectively to and 00000000000 → 00000000000 and 11111111111 → 11111111111, producing no transitions at all. The one-bit transformation vector changes also from 0 → 1 indicating inversion of the entire 11-bit data vector.

This produces new worst case conditions exemplified by, 00000000000 → 00000111111, that is 6 of the 11 bits changing. In these cases, inversion will result in 5 bits changing 00000000000 → 11111000000, and also the changing of the transition vector bit. Thus, the worst case number of bit transitions has been reduced from 11 to 6, a 45.5% reduction.

To determine the average number of logic transitions for random data, a simulation program was written that tested each possible state transition of an 11 bit number. The algorithm described above was then applied and the results observed. The average number of logic transitions per 11-bit data vector without transformation is 5.5, and the average number of logic transitions after transformation is 4.4 for a 21% reduction in the number of logic transitions.

FIGURE 4 is a table of three-bit data vectors showing the number of logic transitions caused by the presentation of subsequent data, whether inversion is warranted, and the number of logic transitions after data inversion. In this example 13 data vectors are propagated through the pipeline (excluding the original state of 000). The average number of transitions for a this particular set of binary coded data is 1.77 logic transformations per vector. The average number of transitions for a three-bit code is 1.5. The average number of transformations fort the transformed data is 1.46 logic transitions per vector for a 17.5% reduction. As the number of bits in the data vector increases, the inclusion of the one-bit translation vector becomes less significant. It is noted that a 50% reduction in transitions is the theoretical limit.

While an odd number of bits in the data vector is preferred, the circuitry of the invention will also work for data vectors having an even number of bits if the control algorithm is modified accordingly. Once such modification is to invert when the number of transitions is equal to, or greater than, (N/2)+1. Thus a 12 bit code would invert when the number of logic transitions is equal to, or greater than 7.

While the invention has been described in the environment of CMOS circuitry, it is applicable in any technology in which power dissipation is proportional to number of bit transitions.

It is herein recognized that controller **330** of FIGURE 3 may be a digital comparator circuit, or may be a function performed by a microprocessor.